Chapter 5

Number Systems

5.1 Data and Information

Data is defined as the collection of facts and figures, whereas the data after processing is called **information.**

Suppose that fifteen students of a class appear in an examination. You are assigned the task of calculating the pass percentage of the overall class and the percentage and grade of each of the students (total masks are 550).

How would you find out the required information? The first step is the data collection; you note down the marks of all students of the class, suppose these are 354, 285, 421, 360, 298, 159, 163, 148, 270, 467, 305, 221, 341, 255, and 311. You are also provided the roll numbers of the students. At this stage the above list of numbers represent **data** you have collected to find the required information. This data is in its raw form; it does not provide the required information. You will have to process it by keeping in mind the required information. Processing may involve certain steps such as applying certain calculations, sorting and formatting etc.

By applying certain calculations we get the following table:

Roll No.	Marks	%age	Grade	Overall pass %age of the class
1	354	64.36	B	
2	285	51.82	C	
3	421	76.55	A	
4	360	65.45	B	
5	298	54.18	C	
6	159	28.91	F	
7	163	29.64	F	
8	148	26.91	F	80%
9	270	49.09	D	
10	467	84.91	A+	
11	305	55.45	C	
12	221	40.18	D	
13	341	62.00	B	
14	255	46.36	D	
15	311	56.55	C	

The above table shows the information extracted from the collected data. It shows clearly, when the data undergoes certain processing we get information. The processing can be done manually or by using a computer. Computer processes data in terms of binary numbers i.e.,0 or 1.

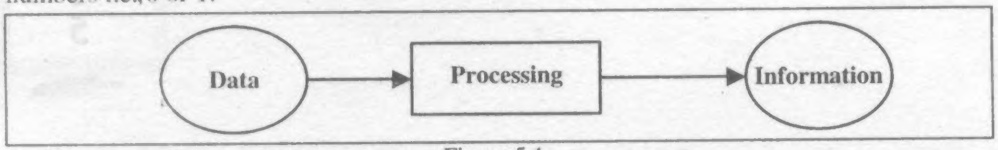

Figure 5.1

Both data and information may be represented in many forms e.g. text, sounds, pictures, graphs, etc. It is useful to observe that a computer is just a data processing machine. It inputs data, perform processing and then output the processed data i.e. information. It is important to know about the different types of data and how these data are represented within a computer.

All computer programs use one or more of the following types of data.

Numeric data, Alphabet data, Alphanumeric data.

5.1.1 Numeric data

Numeric data is used to represent different quantities on which arithmetic is to be performed e.g. marks of different students, sales records of goods at a shop etc. Mostly this data is represented as integers or real numbers e.g. 40, 323, -76.07 etc. there are two types of numeric data:

- Integer
- Real number

5.1.2 Alphabetic data

Alphabet data only consists of a fixed set of alphabetic characters e.g. data consisting of English alphabets A, B, C... Z as well as a, b, c... z. We can use these English alphabets to represent names of students in a class. This data is represented as a sequence of characters and no arithmetic operations can be carried out on it.

5.1.3 Alphanumeric data

Alphanumeric data contains alphabets, numbers and other special characters i.e. $, #, % etc. Example of such data can be telephone numbers and addresses such as, (092) 051-2345682 and House # 967, street 9, ABC colony, Rawalpindi etc.

5.2 Number Systems

A number system defines a set of values used to represent different quantities. For example; we can represent the number of students in our class or number of viewers watching a certain TV program etc. It is interesting to note that in a digital computer all kinds of information and data (e.g. audio, Graphics, Video, Text and Numeric) are represented as binary numbers. In general, decimal system is used while the computers use binary number system. Octal and Hexadecimal numbers systems are also used commonly in computer system that's why, we need a conversion from one number system to another.

5.2.1 Decimal (Base 10) Number System

We are familiar with ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and also know that any value can be represented by using these ten digits only. For example, the value four hundred and fifty three can be written as:

$$453 = 4 \times 10^2 + 5 \times 10^1 + 3 \times 10^0.$$

Do you remember $10^0 = 1$?
What do you know about $N^0 = 1$ where N is not 0?

We can also write numbers with fractional parts in this system by using only ten digits and the decimal point (.) e.g. 139.78 can be written as,

$$139.78 = 1\times10^2 + 3\times10^1 + 9 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$$

It is a positional number system that means the position of a digit within a number is very important so that 39 and 93 represent two different values.

Can you name a non-positional number system?

In this system of numbers each number consists of digits located at different positions with position 0 is the first digit towards the left side of the decimal point, position 1 is the second digit towards the left side of the decimal point and so on. Similarly the first digit towards the right side of the decimal point is at position –1, second has position -2 and so on. Each position has a weight assigned to it e.g. position 1 has weight 10^1 position 2 has weight 10^2 and so on. This is shown in the table below:

Note that positions have weights in powers of 10 because we have 10 digits in the number system

Table demonstrating the position and weights of digits for the number 57231.21

Position	4	3	2	1	0	-1	-2
Face Value	5	7	2	3	1	2	1
Weights	10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}

or, $57231.21 = 5 \times 10^4 + 7 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 1 \times 10^{-2}$

It is clear from the above discussion that the value of the number is determined by multiplying the digits with the weight of their position and adding the results. This method is called the expansion method. The digit at the extreme right side of a number is called the Least Significant Digit (LSD) because it has least weight and the digit at the extreme left side of a number is called the Most Significant Digit (MSD) because it has maximum weight. For example; in the number 724, 7 is most significant and 4 is least significant.

5.2.2 Binary Number System

This number system uses only two digits 0 and 1 to represent any quantity. These digits are called Binary digits or BIT. Like the decimal number system this is also a positional number system and each position has a weight that is a power of 2. For example

Why do we use powers of 2 in the expansion of a binary number?

$$01001_{(2)} = 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = 9_{(10)}$$

The position 0 has weight $2^0 = 1$ and position 1 has weight 2^1. Similarly, we can also represent fractional binary numbers as shown below:

$$101.101_{(2)} = 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = 5.625_{(10)}$$

Table demonstrating the position and weights of digits for the number $101.101_{(2)}$

Position	2	1	0		-1	-2	-3
Face Value	1	0	1	.	1	0	1
Weights	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}

5.2.3 Hexadecimal Number System

You may have observed that binary numbers are not very easy to work with because even very small numbers $277_{(10)}$ need at least 9 bits i.e., $277_{(10)} = 0100010101_{(2)}$ and also converting binary to decimal and decimal to binary needs much calculation. Because of these difficulties computer scientists use another number system frequently. This number system is base 16 or hexadecimal number system .This number system uses sixteen different digits. The digits are:

Remember that
A = Ten
B = Eleven
C = Twelve
D = Thirteen
E = Fourteen
F = Fifteen.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

$758_{(16)}$ is a hexadecimal number and is different from $758_{(10)}$ which is seven hundred and fifty seven. You may read $758_{(16)}$ as **Seven Five Eight Base Sixteen** and not as seven hundred and fifty eight. A fractional hexadecimal number e.g., $758.D1_{(16)}$ can be represented as shown below:

$$758.D1_{(16)} = 7\times16^2 + 5\times16^1 + 8\times16^0 + D\times16^{-1} + 1\times16^{-2} = 1880.8164_{(10)}$$

(up to four decimal places)

Table demonstrating the position and weights of digits for the number $758.D1_{(16)}$

Position	2	1	0		-1	-2
Face Value	7	5	8	.	D	1
Weights	16^2	16^1	16^0		16^{-1}	16^{-2}

5.2.4 Octal Number System

This is another number system that can also be used in computers. This is the base-8 number system or octal number system. This number system uses only eight different digits to represent numbers. The digits are 0, 1, 2, 3, 4, 5, 6, 7. In this number system $751_{(8)}$ is a valid number and is different from seven hundred and fifty one. Clearly **821** can not be a number in this system as 8 is not a valid digit in this number system. An octal number e.g., $630.4_{(8)}$ can be represented as shown below:

$$630.4_{(8)} = 6\times8^2 + 3\times8^1 + 0\times8^0 + 4\times8^{-1} = 408.5_{(10)}$$

Table demonstrating the position and weights of digits for the number $630.4_{(8)}$

Position	2	1	0		-1
Face Value	6	3	0	.	4
Weights	8^2	8^1	8^0		8^{-1}

5.3 Number System Conversion

We use decimal system and computer use binary system. Also the octal and hexadecimal number systems are frequently used in computerized data processing system. An interesting problem is the conversion of data from one number system to another number system.

5.3.1 Conversion of Decimal Number into Binary

To convert a decimal number into Binary, we can use the repeated division procedure as shown in following example.

Example: Convert 27 into binary

Solution:

	Number	Remainder
2	27	
2	13	1
2	6	1
2	3	0
2	1	1
	0	1

We should stop the division process when 0 is the answer of some division and collect the remainders in reverse order as is indicated by the arrow:

So $27_{(10)} = 011011_{(2)}$

5.3.2 Converting Fractional Decimal Numbers to Binary

Following example demonstrates the process of converting a decimal fraction in binary:

Example: Convert 0.56 into binary. Give answer up to 6 decimals

Solution:

			Result	Fractional part	Integral part
2	×	0.56	1.12	12	1
2	×	0.12	0.24	24	0
2	×	0.24	0.48	48	0
2	×	0.48	0.96	96	0
2	×	0.96	1.92	92	1
2	×	0.92	1.84	84	1
2	×	0.84	1.68	68	1
2	×	0.68	1.36	36	1

$0.56_{(10)} = 0.100011_{(2)}$

The bits to the left of point are obtained by using the integer parts of the answers in the order shown by the arrow. Note that we continue this process of multiplication until we get 0 or a repeated value as answer or get required number of bits.

5.3.3 Converting Real Numbers into Binary

Next example shows the process of converting a real number (i.e., a number that has both an integer part and a fractional part) into binary.

Example: Convert $56.25_{(10)}$ into binary

Solution: To convert this real number we independently convert the integral part (i.e, 56) and the fractional part (i.e.,0.25) into binary by using the processes given above

	Number	Remainder
2	56	
2	28	0
2	14	0
2	7	0
2	3	1
2	1	1
	0	1

$56 = 0111000_{(2)}$

			Result	Fractional part	Integral part
2	×	0.25	0.5	5	0
2	×	0.5	1.0	0	1

$0.25 = .01_{(2)}$

so, **$56.25 = 0111000.01_{(2)}$**

Note: The above result is obtained by connecting the two results.

5.3.4 Converting Binary Numbers into Decimal

As is shown in the examples below we can use the expansion method to convert a binary number into decimal system.

Example 1: Convert $011011_{(2)}$ into decimal

Solution: $011011_{(2)} = 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 27_{(10)}$

Example 2: Convert $1110.11_{(2)}$ into decimal

Solution: $1110.11_{(2)} = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$= 8 + 4 + 2 + 0 + 1/2 + 1/4 = 14.75$

5.3.5 Conversion of Decimal into Hexadecimal

The process of converting a decimal number into hexadecimal is shown in the examples below.

Example 1: Convert $185_{(10)}$ into hexadecimal

Solution:

	Number	Remainder
16	185	
16	11	9
	0	B

So **$185_{(10)} = 0B9_{(16)}$**

Example 2: Convert $0.3_{(10)}$ into hexadecimal.

Solution:

			Result	Fractional part	Integral part
16	×	0.3	4.8	8	4
16	×	0.8	12.8	8	12=C
16	×	0.8	12.8	8	12=C

So **$0.3_{(10)} = 0.4C_{(16)}$** (because C is the repeating value therefore, as a Convention, we shall take it once only)

Example 3: Convert $185.3_{(10)}$ into hexadecimal

Solution: As given in the above two examples

$185_{(10)} = B9_{(16)}$ and $0.3_{(10)} = 0.4C_{(16)}$

So $185.3 = 0B9.4C_{(16)}$

As you can see the conversion of a decimal to hexadecimal is similar to the conversion of decimal into Binary. Actually this repeated division or repeated multiplication method can be used for converting a decimal number into any other base .

5.3.6 Conversion of Hexadecimal into Decimal

We can also convert a hexadecimal number into decimal by using the expansion method shown in the following examples :

Example 1: Convert 0B9($_{(16)}$ into decimal

Solution: $0B9_{(16)} = 0 \times 16^2 + B \times 16^1 + 9 \times 16^0 = 0 \times 16^2 + 11 \times 16^1 + 9 \times 16^0 = 185_{(10)}$

Example 2: Convert $0B9.4C_{(16)}$ into decimal

Solution:

$$\begin{aligned} 0B9.4C_{(16)} &= 0 \times 16^2 + B \times 16^1 + 9 \times 16^0 + 4 \times 16^{-1} + C \times 16^{-2} \\ &= 0 \times 16^2 + 11 \times 16^1 + 9 \times 16^0 + 4 \times 16^{-1} + 12 \times 16^{-2} \\ &= 0 + 176 + 9 + 4/16 + 12/256 \\ &= 0 + 176 + 9 + 1/4 + 3/64 = 185.296875_{(10)} \end{aligned}$$

5.3.7 Conversion of Hexadecimal into Binary

As was stated before that the conversion between binary and decimal numbers is a tedious process. On the other hand the conversion of a hexadecimal to Binary is very easy and is shown in the example below.

Example 1: Convert $10A8_{(16)}$ into Binary

Solution:

Step 1: Convert each digit into binary separately and write it in 4 bits.

$1 = 0001_{(2)}$

$0 = 0000_{(2)}$

$A = 1010_{(2)}$

$8 = 1000_{(2)}$

Step 2: Replace the digits of the hexadecimal number with the four bits obtained in step 1.

$10A8_{(16)} = 0001\ 0000\ 1010\ 1000_{(2)}$

Example 2: Convert $A1.03_{(16)}$ into Binary

Solution:

Step 1: Convert each digit independently into binary and write it in 4 bits.

$A = 1010_{(2)}$
$1 = 0001_{(2)}$
$0 = 0000_{(2)}$
$3 = 0011_{(2)}$

Step 2: Replace the digits of the hexadecimal number with the four bits obtained in step 1.

$10A8_{(16)} = 1010\ 0001\ .0000\ 0011_{(2)}$

5.3.8 Conversion of Binary into Hexadecimal

The conversion of binary to hexadecimal is the reverse of conversion of hexadecimal into binary and is shown in the example below.

Example 1: Convert $10010011_{(2)}$ into hexadecimal.
Solution:

Step 1: First divide your number into groups of 4 bits starting from the right side

$10010011_{(2)}$ is divided into the following two groups 1001 and 0011

Step 2: Convert each group into hexadecimal.

$1001=9_{(16)}$ and $0011=3_{(16)}$

Step 3: Replace each group by its hexadecimal equivalent.

$1001\ 0011_{(2)} = 93_{(16)}$

Example 2: Convert $101100.1_{(2)}$ into hexadecimal.

Solution:

Step 1: First divide your number into groups of 4 bits starting from the decimal point position so $101100.1_{(2)}$ is divided into the following three groups 1100 and 0010 on the left of decimal point and 1000 on the right of decimal point

Step 2: Convert each group into hexadecimal.

$1100 = 12 = C_{(16)}$, $0010 = 2_{(16)}$ and $1000=8_{(16)}$

Explain!
Why do we make 4 bit groups?

Step 3: Replace each group by its hexadecimal equivalent.

$10\ 11\ 00.1_{(2)} = 2C.8_{(16)}$

Note that if the last group on the left of decimal has less than 4 bits then extra 0 bits are added at the left end of the number. Similarly if the last group on the right of decimal has less than 4 bits then extra 0 bits are added at the right end.

The following table can be used to convert any hexadecimal number into binary

Table for Hexadecimal to Binary Conversion

Hexadecimal Number	Binary Equivalent	Hexadecimal Number	Binary Equivalent
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

5.3.9 Conversion of Decimal into Octal

The repeated division method will be used to convert the decimal number into octal and is shown in the following example.

Example 1: Convert $185_{(10)}$ into Octal
Solution:

		R
8	185	
8	23	1
8	2	7
8	0	2

So $185_{(10)} = 0271_{(8)}$

We can also convert a fractional octal number into decimal by using the expansion method shown in the next example

Example 2: Convert $0.3_{(10)}$ into octal and give answer up to 5 decimal places.

Solution:

$8 \times 0.3 = 2.4$	0.4	2
$8 \times 0.4 = 3.2$	0.2	3
$8 \times 0.2 = 1.6$	0.6	1
$8 \times 0.6 = 4.8$	0.8	4
$8 \times 0.8 = 6.4$	0.4	6

So $0.3_{(10)} = 0.23146_{(8)}$

Example 3: Convert $185.3_{(10)}$ into octal . Give answer up to 5 decimal places.
Solution: As given in the two examples above

$185_{(10)} = 0271_{(8)}$ and $0.3_{(10)} = 0.23146_{(16)}$

So $185.3 = 0271.23146_{(8)}$

5.3.10 Conversion of Octal into Decimal

Example 1: Convert $0271_{(8)}$ into decimal

Solution: $0271_{(8)} = 0 \times 8^3 + 2 \times 8^2 + 7 \times 8^1 + 1 \times 8^0 = 185_{(10)}$

Example 2: Convert $271.231_{(8)}$ into decimal

Solution: $0271.231_{(8)} = 0\times 8^3 + 2\times 8^2 + 7\times 8^1 + 1\times 8^0 + 2\times 8^{-1} + 3\times 8^{-2} + 1\times 8^{-3}$

$= 0 + 128 + 56 + 1 \quad + 2/8 + 3/64 + 1/512$

$= 185.2988_{(10)}$

5.3.11 Conversion of Octal Number into Binary

The conversion of an octal number to binary is similar to the conversion of hexadecimal into binary and shown in the following example.

Example 1: Convert $107_{(8)}$ into Binary

Step 1: Convert each digit independently into binary and write it in 3 bits.

$1 = 001_{(2)}$
$0 = 000_{(2)}$
$7 = 111_{(2)}$

Step 2: Replace the digits of the hexadecimal number with the three bits obtainedin step 1.

$107_{(8)} = 001\ 000\ 111_{(2)}$

Example 2: Convert $107.52_{(8)}$ into Binary

Solution:

Step 1: Convert each digit independently into binary and write it in 3 bits.

$1 = 001_{(2)}$ $0 = 000_{(2)}$ $7 = 111_{(2)}$
$5 = 101_{(2)}$ $2 = 010_{(2)}$

Step 2: Replace the digits of the octal number with the three bits obtained in step 1

$107.52_{(8)} = 001\ 000\ 111.\ 101\ 010_{(2)}$

5.3.12 Conversion of Binary into Octal

The conversion of binary number to octal can be done directly and is shown in the example below

Example 1: Convert $10010011_{(2)}$ into octal

Solution:

Step 1: First divide your number into groups of 3 bits starting from the right side

So $10010011_{(2)}$ is divided into the following three groups

010, 010 and 011

Step 2: Convert each group into octal.

$010=2_{(8)};$ $010=2_{(8)};$ $011=3_{(8)}$

Step 3: Replace each group by its octal equivalent.

$010\ 010\ 011_{(2)} = 223_{(8)}$

Example 2: Convert $11010.11_{(2)}$ into octal

Solution:

Step 1: First divide your number into groups of 3 bits starting from the right side

So $11010.11_{(2)}$ is divided into the following three groups 010, 011 on the left of decimal and 110 on the right of decimal

Can you guess the reason of making groups of 3 Bits?

Step 2: Convert each group into octal.

$010=2_{(8)}$, $011=3_{(8)}$, $110=6_{(8)}$

Step 3: Replace each group by its octal equivalent.

$100100.110_{(2)} = 32.6_{(8)}$

Note: If the last group has less than 3 bits then add extra 0 bits at the left or right end depending if it is on the left or right side of the decimal point, respectively.

Table for Octal to Binary Conversion

Octal Number	Binary Equivalent	Octal Number	Binary Equivalent
0	000	4	100
1	001	5	101
2	010	6	110
3	011	7	111

5.4 Representation of Numbers using 1's and 2's Complements

Representing Signed Numbers

Now that you know how to represent positive numbers in different number systems for example. Base 2, Base 8, Base 10 and Base 16. It is time to look at another interesting question.

How do we represent both positive and negative numbers in Binary number system?

There are many methods of representing signed numbers in binary e.g. **sign-magnitude** method, **1's complement** method, **2's complement** method and **Access notation**. In this section we will study 1's complement method and 2's complement methods of representing signed numbers. These two methods are very useful when we want to perform binary arithmetic.

5.4.1 1's Complement Method

To understand this method of representing signed numbers we first see the meaning of the 1's complement of a binary number.

1's complement of an 8-bit binary number is obtained by subtracting the number from $11111111_{(2)}$ as shown in the example below.

Example 1: **Take 1's complement of the binary number 10011001.**

Solution:

$$\begin{array}{lr} & 11111111 \\ & -\,10011001 \\ \hline \textbf{1's Complement} & 01100110 \end{array}$$

Observe that 1's complement of a binary number can be directly obtained by changing all 0's to 1's and all 1's to 0's

Example 2: Take 1's complement of the binary number 01100110 directly.

Solution:

Original number	01100110
1's Complement	10011001

Representation of Negative numbers using 1's complement:

To represent the negative numbers in 1's complement form, we perform the following steps:

- First determine the number of bits to represent the number.
- Convert the modules of the given number in binary.
- Place a 0 in MSB and binary conversion of the number in remaining bits MSB (Most Significant Bit).
- Take 1's complement of the result.

Example 3: Represent -$54_{(10)}$ in 1's complement form using 8 bits.
Solution:

Number of bits	= 8
$54_{(10)}$	$0110110_{(2)}$=$00110110_{(2)}$(in 8 bits)

So-54 in 1's complement form = 11001001

So 1's complement representation of negative integers is same as the usual binary mber representation and will always have a 0 in MSB..

5.4.2 2's Complement Method

We know that most computers use 16 bits to represent integers. When numbers are represented in fixed number of bits the 2's complement method is a very useful way of representing signed number. Most computers represent integers using this method. Many digital calculators also used this method for representing integers.

2's complement of a binary number can be obtained by first taking 1's complement and then adding 1 in the result. This process is shown in the example below:

Example 1: Take 2's complement of the binary number $01100110_{(2)}$.

Solution:

Step 1: Taking 1's complement of the given number results in 10011001.

Step 2: Adding 1 in the result give us

```
      10011001
            +1
      --------
      10011010
```

2's complement of 01100110 = 10011010

We can obtain 2's complement of a binary number directly without taking 1's complement. To take 2's complement directly copy the number without any change up to the first (least significant) 1 in the number and change reaming 0s to 1s and 1s to 0s. This process is shown in the example below.

Example 2: Take 2's complement of the binary number $01100110_{(2)}$ directly.

Given number:	01100110
2's Complement:	10011010

Representation of Negative numbers using 2's complement

To represent the negative numbers in 2's complement form we perform the following steps:

- First determine the number of bits to represent the number
- Convert the modules of the given number in binary.

- Place a 0 in MSB and binary of the number in remaining bits.
- Take 2's complement of the result.

This is shown in the example given below.

Example 3: Represent -54$_{(10)}$ in 2's complement form using 8 bits.

Solution:

Number of bits	= 8
Modulus of –54= 54	= 0110110
54 in 2's complement form	= 00110110
-54 in 2's complement form	= 11001010

Clearly 2's complement representation of a negative integer will alway have a 1 in the most significant bit. The smallest integer in 2's complement form using 8 bits is 10000000 = -128 = -2^7 and the largest integer in 2's complement form using n bits is $2^{(n-1)}$

5.5 Binary Arithmetic

In this section we are going to learn the basic arithmetic operations i.e. addition, subtraction, multiplication and division for binary integers. Following table shows the results of performing the addition operations on two **bits.** This table can also be used to add two multi-bit binary numbers.

5.5.1 Binary Addition

Following table shows the results of performing the addition operations on two **bits.** This table can also be used to add two multi-bit binary numbers.

Operation	Result
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	0 **with a carry of 1**

The following example shows the process of adding two binary numbers by using the results from the table above.

Example: Add 01011101$_{(2)}$ and 00110010$_{(2)}$.

Solution:

```
Carry     1 1 1
          0 1 0 1 1 1 0 1
    +     0 0 1 1 0 0 1 0
          ---------------
          1 0 0 0 1 1 1 1
```

Note that 1+1 should be 2. But in binary system 1+1 is 0 and a carry of one is generated.

Challenge

Can you guess 1+1+1 =?

Clearly the process of adding two binary numbers is similar to that of adding two decimal numbers. But in this process we use the rules given in the table above.

5.5.2 Binary Subtraction

The process of subtracting two binary numbers is also similar to decimal numbers subtraction. Following table shows the rules of subtracting two bits.

Operation	Result
0 – 0	0
0 – 1	1 with a barrow from the next position
1 – 0	1
1 – 1	0

Now that you know the rules for subtracting two bits, let us learn the process of subtracting two binary numbers through the following example

Example: Subtract $01011101_{(2)}$ from $10110010_{(2)}$.

> **Observation**
> By using 1's and 2's complement method, we can perform subtraction by using addition only.

Solution:

```
Barrow        10              10
             ↗               ↗
            1  0  1  1  1  1  1  0
  –         0  1  0  1  1  1  0  1
           ------------------------
            0  1  1  0  0  0  0  1
```

Note that when you take a barrow from the higher position 0 becomes $10_{(2)}$ and $10_{(2)} - 1_{(2)} = 1$

As it is difficult and expensive to make a computer that uses this method for performing subtraction. Most day computers use 2's complement or 1's complement method for performing subtraction.

Subtraction Using 1's Complement

The following examples explain fully the process of subtraction using 8-bit 1's complement method .

Example 1: Calculate 38 – 29 using 8-bit 1's complement method .

Solution: Here 38 – 29 = 38 + (-29)

Step 1: Write magnitude of both the numbers in binary form using 8 bits.

$38 = 00100110_{(2)}$ and $29 = 00011101_{(2)}$

Step 2: Represent negative numbers in 1's complement.

- 29 = 11100010

Step 3: Add the 1's complement representation .

```
                         00100110
                 +       11100010
                 ----------------
End  carry: 1            00001000
Add End carry: +                1
                 ----------------
Answer:                  00001001
```

So the answer is 00001001 and it is in 1's complement.

Step 4: Convert the 1's complement result into decimal

00001001 = 9

Example 2: Calculate 45 – 63 using 8-bit 1's complement method .

Solution:

We can write 45 – 63 = 45 + (-63)

Step 1: Write magnitude of both the numbers in 8 bits.

$45 = 00101101_{(2)}$ and $63 = 00111111_{(2)}$

Step 2: Represent number in 1's complement

- 63 = 11000000

Step 3: Add the 1's complement representation

```
              00101101
        +     11000000
      ------------------
End Carry: 0      11101101
Add  End Carry: +        0
      ------------------
Answer:           11101101
```

So the answer is 1 1 1 0 1 1 0 1 and it is in 1's complement

Step 4: Convert the 1's complement result into decimal

11101101 = - 00010010 = -18

Note: If the End carry in the sum is 0 then we do not need to perform step 4.

Example 3: Calculate (– 54 – 30) using 8 bits 1's complement method.

Solution: (First number is negative as well)

We can write -54 – 30 = (-54) + (-30)

Step 1: Write magnitude of both the numbers in 8 bits.

$54 = 00110110_{(2)}$ and $30 = 00011110_{(2)}$

Step 2: Represent both numbers in 1's complement form.

-54 = 11001001 and 30 = 11100001

Step 3: Add the 1's complement representation.

```
              11001001
        +     11100001
      ------------------
End Carry:1       10101010
Add End Carry: +         1
      ------------------
Answer:           10101011
```

So the answer is 1 0 1 0 1 0 1 1 and it is in 1's complement.

Step 4: Convert the 1's complement result into decimal.

As the MSB is 1 so it is a negative number so,

10101011 = -01010100 = -84

Note: 1's complement use addition operation twice for subtracting binary numbers. First for adding the numbers and then for adding the final carry.

Subtraction Using 2's Complement.

The process of subtracting two binary numbers using 2's complement is explained through the following examples. We are using the same questions given in the examples of 1's complement method to so that you can compare the two methods.

Example 1: Calculate 38 – 29 using 8-bit 2's complement method

Solution: Here 38 – 29 = 38 + (-29)

Step 1: write magnitude of both numbers in 8 bits.

38 = 00100110$_{(2)}$ and 29 = 00011101$_{(2)}$

Step 2: represent negative number in 2's complement.

- 29 = 11100011

Step 3: add the 2's complement representation and ignore the end carry.

```
                 00100110
            +    11100011
            -------------
End Carry 1      00001001
```

So the answer is 0 0 0 0 1 0 0 1 and it is in 2's complement.

Step 4: Convert the 2's complement result into decimal

00001001 = 9

In the following examples we are going to subtract a larger number from a smaller number.

Example 2: Calculate 45 – 63 using 8-bit 2's complement method

Solution:

We can write 45 – 63 = 45 + (-63)

Step 1: write magnitude of both the numbers in 8 bits.

45 = 00101101$_{(2)}$ and 63 = 00111111$_{(2)}$

Step 2: represent negative numbers in 2's complement.

- 63 = 11000001

Step 3: add the 2's complement representation and ignore the end carry.

```
                 00101101
            +    11000001
            -------------
End Carry 0      11101110
```

So the answer is 1 1 1 0 1 1 1 0 and it is in 2's complement form.

Step 4: Convert the 2's complement result into decimal

11101110 = - 00010010 = -18

Note that when we calculate –97 – 85 using 8-bit 2's complement, the answer should be –182 but it comes out to be 74. Can you explain why?

Example 3: Calculate (-54 – 30) using 8 bits 2's complement method.

Solution: We can write -54 – 30 = (-54) + (-30)

Step 1: write magnitude of both numbers in 8 bits.

54 = 00110110$_{(2)}$ and 30 = 00011110$_{(2)}$

Step 2: represent both numbers in 2's complement.

-54 = 11001010 and -30 = 11100010

Step 3: add the 2's complement representation and ignore the end carry.

```
                 11001010
            +    11100010
            -------------
End Carry 1      10101100
```

So the answer is 1 0 1 0 1 1 0 0 and it is in 2's complement.

Step 4: Convert the 2's complement result into decimal

10101100 = - 01010100 = -84

You may have noticed that by using 1's complement and 2's complement method we can perform subtraction by using the addition operation only. So if some one the same makes a digital circuit for adding two binary numbers we can also subtract the binary numbers by using circuit and also if some digital computer can add two binary numbers, then it can subtract the numbers as well.

5.5.3 Binary Multiplication

In this section we will first learn how to multiply two unsigned binary integers by using the familiar multiplication process and then also see another very interesting way of multiplication.

Following table gives the basic multiplication rules for two bits

. confuses with the decimal point so, replace it with x.

Product	Answer
0.0	0
0.1	0
1.0	0
1.1	1

The following example describe the process of multiplying two 4-bit binary numbers.

Example: Calculate $0110_{(2)} \times 1011_{(2)}$

Solution:

```
Multiplicand:          0 1 1 0
Multiplier:            1 0 1 1
              ------------------
                       0 1 1 0
                     0 1 1 0 ×
                   0 0 0 0 × ×
                 0 1 1 0 × × ×
              ------------------
                 1 0 0 0 0 1 0
```

Obviously this is the usual way of multiplying numbers.

5.5.4 Division of Binary Numbers

Binary division method is demonstrated in the following examples.

Example 1: Calculate $01001101_{(2)} \div 111_{(2)}$

Solution:

```
           0 1 0 1 1
      ----------------
1 1 1 |  0 1 0 0 1 1 0 1
      |  0 0 1 1 1
      ----  -
         ------------
             1 0 1 0
             0 1 1 1
          -
         ------------
               1 1 1
               1 1 1
            -
         ------------
             0 0 0
```

You can easily verify that $01001101_{(2)} = 77_{(10)}$, $111_{(2)} = 7_{(10)}$, and $01011_{(2)} = 11_{(10)}$

Example 2: Calculate $01111001_{(2)} \div 1011_{(2)}$

Solution:

```
              0 1 0 1 1
       1011 | 0 1 1 1 1 0 0 1
              _0 1 0 1 1
              ----------------
                1 0 0 0 0
               _1 0 1 1
              ----------------
                  1 0 1 1
                 _1 0 1 1
              ----------------
                  0 0 0 0
```

You can easily verify that $01111001_{(2)} = 121_{(10)}$, $1011_{(2)} = 11_{(10)}$, and $01011_{(2)} = 11_{(10)}$

5.6 Fixed Point and Floating Point Number Representations

5.6.1 Fixed Point Representation

To learn how computers represent real numbers using the fixed-point representation of numbers. We will first learn the concept using the decimal number system.

Suppose you have been told to write all real numbers by using the following rule.

The number will always have exactly four digits before the decimal point and exactly three digits after the decimal point.

Second column of the following table shows how different numbers will be written by using this rule.

Number	Number Written By Using the rule	Number without decimal point
73.4	0073.400	0073400
120.3456	0120.345 **(6 can not be written)**	0120345
110	0110.000	0110000
11101.0	1101.0000 **(so can not be represented)**	11010000

These numbers can be called fixed-point numbers because the position of the decimal point is fixed within the number. If numbers are written by using this format, we do not need to write the decimal point as it is always on the left of right digit from the left-hand side. This is shown in the third column of the table.

It is also clear from the table above that the numbers having more than 4-digit integeral part or more than 3-digit fractional part can not be represented accurately using this representation.

Computers are made to represent real numbers in a similar way. To represent a real number in a computer it may be programmed to use the following rules.

- Numbers may be represented using 8 bits, 16 bits, 32 bits or more.
- Decimal point will not be written.
- Decimal point is always after the 10th bit.
- MSB is used to represent sign of the number (0 means +ve and 1 means –ve).
- Next 9 bits will be used to store the integeral part of the number.
- Remaining 6 bits will be used to represent the fractional part of the number

This format is shown below:

Sign Bit	Integral part of the number									Fractional Part					

This representation of real numbers is called the fixed-point representation. Following table shows how some of the binary numbers that can be represented using this representation.

Decimal Number	Binary Number	Number in Fixed-Point Form
3.625	011.1010	0000000011101000
247.90625	11110111.11101	0011110111111010
-7.66796875	-0111.10101011	1000000111101010 **(remaining bits do not fit)**
-81.765625	-1010001.110001	1001010001110001

Advantage of this representation is that it is very simple to use and the disadvantage is that very small or very large numbers cannot be represented using it.

Following examples show the process of representing different numbers in this form.

Example 1: Represent 23.6 in 16-bit fixed-point form using 10 bits for integral part.
Solution:

$23 = 010111_{(2)}$ and $0.6 = .1001001$

$23.6 = 010111.1001001 = 0000010111.100100$

and in fixed point form $23.6 = 0000010111100100$

Example 2: Represent-36.25 in 16-bit fixed-point form using 10 bits for integral part.
Solution:

$36 = 0100100_{(2)}$ and $0.25 = .01$

$36.25 = 0100100.01 = 0000100100.01$

And in fixed point form $-36.25 = 1000100100010000$

The following examples show the conversion of fixed-point numbers into decimals.

Example 3: Convert the 16-bit fixed-point number 0100010111100100 **into decimal where 10 bits are used for integral part**
Solution:
Integral part =0100010111 and **Fractional Part** = .100100

Now $0100010111_{(2)} = 279$

$.100100_{(2)} = 0.5 + .0625 = 0.5625$

So $0100010111100100 = 279.5625_{(10)}$

Example 4: Convert the 16-bit fixed-point number 1 0 0 0 1 1 0 1 1 1.1 1 0 0 0 0 into decimal where 10 bits are used for integer part .

Solution:

The Integral part = 1 0 0 0 1 1 0 1 1 1 and **Fractional Part** = 1 1 0 0 0 0

Now 1 0 0 0 1 1 0 1 1 1 $_{(2)}$ = -55

And . 1 1 0 0 0 0 $_{(2)}$ = 0.5 + .25 = 0.75

So 0 1 0 0 0 1 0 1 1 1 1 0 0 1 0 0 =-55.75$_{(10)}$

5.6.2 Floating–Point Representation

Floating point representation is another useful way of writing real numbers. In this format, very small and very large numbers can be represented efficiently. Before learning the floating-point representation of binary numbers let us consider the following format of writing the decimal in the notation:

$$174.592 = 0.174592 \times 10^3.$$

This representation is known as the scientific notation. In this notation,10 is the base, power of 10 is called the **exponent** and the **number** is called the **mantissa** so in the above number base is 10, mantissa is 0.174592 and the exponent is 3.

We can also write binary numbers in a similar way. For example, the number 1000.1101 = 0.10001101 × 2^4 where in this case the base is 2, the mantissa is**10001101** and the exponent is 4. Now consider the following way of writing the binary numbers .

Write the sign	Write the Exponent	Write the Mantissa

For example the table given below shows different binary numbers written in this format. Note that the binary point is adjusted such that leading bit in the mantissa of all numbers is always 1.

Number	Sign	Exponent	Mantissa
1.10001101×2^4	+	4	1.10001101
$- 1.1101101 \times 2^5$	-	5	1.1101101
$1101.0011 = 1.1010011 \times 2^3$	+	3	1.1010011
$0.011011 = 1.1011 \times 2^{-2}$	+	-2	1.1011

This format for writing real numbers is called the floating point representation. Most digital computers use this format for representing real numbers. In this book we will use the following floating-point format:

S	6-bit Exponent						9-bit Mantissa								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

So the computers use 16 bits for representing floating–point numbers. The MSB marked as **S** is used to represent the sign of the number. Next 6 bits are used to represent the exponent and the remaining 9 bits are used to represent mantissa of the number. Following are some important points about this representation:

- The **sign** of a binary floating-point number is represented by a single bit. A 1 bit indicates a negative number, and a 0 bit indicates a positive number.
- The **exponent** is a signed integral and is represented as 6-bit 2's complement number.
- As first bit of the **mantissa** is always 1, so in most modern computers it is not written.

Following three examples show the process of representing numbers in floating–point format.

Example 1: Represent 17.5 as a 16-bit floating point number.

Solution:

Step 1: convert the number into binary

$$17.5 = 0\,1\,0\,0\,0\,1\,.\,1\,0_{(2)} = 1\,.\,0\,0\,1\,1\,0 \times 2^4$$

Step 2: Represent the number in floating–point format.

Sign = + = 0

Exponent = 4 and in 6-bit 2'scomplement form 0 0 0 1 0 0

Mantissa = 1 . 0 0 1 1 0 = 1 . 0 0 1 1 0 0 0 0 0

So the number in floating–point format is

S	6-bit Exponent						9-bit Mantissa								
0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0

Note : The first 1 and the binary point in the mantissa is not written.

Example 2: Represent -117.125 as a 16-bit floating point number.

Solution:

Step 1: convert the number into binary

$$-117.125 = -0\,1\,1\,1\,0\,1\,0\,1\,.\,0\,0\,1\,0_{(2)} = -1\,.\,1\,1\,0\,1\,0\,1\,0\,0\,1\,0 \times 2^6$$

Step 2: represent the number in floating point format

Sign = - = 1

Exponent = 6 and in 6-bit 2's complement form 0 0 0 1 1 0

Mantissa = 1 . 1 1 0 1 0 1 0 0 1 0 = 1 . 1 1 0 1 0 1 0 0 1

So the number in floating point format is

S	6-bit Exponent						9-bit Mantissa								
1	0	0	0	1	1	0	1	1	0	1	0	1	0	0	1

Example 3: Represent– 0 . 0 0 0 1 1 0 1 0 0 1 0 0 1 $_{(2)}$ as a 16-bit floating point number.

Solution:

-0 . 0 0 0 1 1 0 1 0 0 1 0 0 1 = -1 . 1 0 1 0 0 1 0 0 1 × 2^{-4}

Sign = - = 1

Exponent = -4 and in 6-bit 2's complement form 1 1 1 1 0 0

Mantissa = 1 1 0 1 0 0 1 0 0 1 = 1 . 1 0 1 0 0 1 0 0 1 0

So the number in floating point format is

S	6-bit Exponent						9-bit Mantissa								
1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	1

Following examples show the process of converting a floating-point number into binary.

Example 4: Convert the following 16-bit floating point numbers into binary.

S	6-bit Exponent						9-bit Mantissa								
0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	0

Solution:

S = 0 = +ve

Exponent = 0 1 1 1 1 0 = 30

Mantissa = 1 . 1 1 0 0 0 1 0 0 0

So the number is 1 . 1 1 0 0 0 1 0 0 0 × 2^{30}

Note that the first 1 in the mantissa is written because it was ignored when the number was written in floating point form.

Example 5: Convert the following 16-bit floating point numbers into binary

S	6-bit Exponent						9-bit Mantissa								
1	1	0	0	1	1	1	0	1	1	1	0	1	1	0	1

Solution:

S = 1 = -ve

Exponent =1 0 0 1 1 1 = - 0 1 1 0 0 1 = -25

Mantissa = 1 . 0 1 1 1 0 1 1 0 1

So the number is -1 0 1 1 1 0 1 1 0 1 × 2^{25}

5.7 Computer Code

We know that alphabetic data consists of only characters and alphanumeric data consists of characters and numbers digits. To represent this data in a computer system we assign a numeric code to each character of the alphabet. For example, we may assign the following codes to different characters e.g. A:65, B:66 etc. Therefore, we can represent both alphabetic and alphanumeric data in a computer system by using these codes.

5.7.1 ASCII (American Standard Code for Information Interchange)

ASCII is one such coding scheme published by ISO (International Standards Organization). It is a 7-bit coding scheme. The codes assigned to various characters are shown in the table below.

TABLE OF ASCII CODES

Code	Character	Code	Character	Code	Character	Code	Character
0		32	Space	64	@	96	
1		33	!	65	A	97	a
2		34	"	66	B	98	b
3		35	#	67	C	99	c
4		36	$	68	D	100	d
5		37	%	69	E	101	e
6		38	&	70	F	102	f
7		39	'	71	G	103	g
8		40	(	72	H	104	h
9		41	)	73	I	105	i
10		42	*	74	J	106	j
11		43	+	75	K	107	k
12		44	,	76	L	108	l
13		45	-	77	M	109	m
14		46	.	78	N	110	n
15		47	/	79	O	111	o
16		48	0	80	P	112	p
17		49	1	81	Q	113	q
18		50	2	82	R	114	r
19		51	3	83	S	115	s
20		52	4	84	T	116	t
21		53	5	85	U	117	u
22		54	6	86	V	118	v
23		55	7	87	W	119	w
24		56	8	88	X	120	x
25		57	9	89	Y	121	y
26		58	:	90	Z	122	z
27		59	;	91	[	123	{
28		60	<	92	\	124	|
29		61	=	93	]	125	}
30		62	>	94	^	126	~
31		63	?	95		127	

Note: There is no character corresponding to the codes 0 – 31 .

Note that the character **A** and **a** have different codes in ASCII. Most computers also use 8-bit ASCII codes. In the 8-bit ASCII codes the remaining 128 codes are used to represent graphical and other special characters.

Following examples demonstrate the use of ASCII to represent different messages.

Example 1: Represent the following message "Binary" using ASCII codes.

Solution: Using the ASCII table we see that

Character	Decimal Code	Binary Code
B	66	01000010
i	105	01101001
n	110	01101110
a	97	01100001
r	114	01110010
y	121	01111001

So we can write
"Binary" as 01000010 01101001 01101110 01100001 01110010 01111001

Example 2: Convert the following ASCII message into English

01010111 01101000 01100001 01110100 00111111

Solution: Using the ASCII table we see that

Binary Code	Decimal Code	Character
01010111	87	W
01101000	104	h
01100001	97	a
01110100	116	t
00111111	63	?

So,we can write
01010111 01101000 01100001 01110100 00111111 to represent the message **What?**

5.7.2 Binary Coded Decimal (BCD)

This coding scheme is used to represent numeric data. We know that decimal numbers system has ten different digits. To represent these digits we need a 4-bit code. In BCD the digits are assigned the following codes.

Table of BCD Codes

Digit	Code	Digit	Code	Digit	Code	Digit	Code	Digit	Code
0	0000	1	0001	2	0010	3	0011	4	0100
5	0101	6	0110	7	0111	8	1000	9	1001

Following example shows the representation of non-negative integers in BCD.

Example: Represent 9807 in BCD.
Solution:
We know that in BCD 9 = 1001 , 8 = 1000 , 0 = 0000 and 7 = 0111
Thus 9807 = 1001 1000 0000 0111

Clearly we need 16 bits to represent this 4-digit number. The same number can be represented in binary by using 14 bits. So the BCD codes use more bits hence require more computer memory. When arithmetic is to be performed on the numbers coded in BCD either they are first converted into binary and then arithmetic is performed or special circuits are designed for this purpose.

5.7.3 Extended Binary Coded Decimal Interchange Code (EBCDIC)

IBM introduced a new way of character coding scheme called EBCDIC (Extended Binary Coded Decimal Interchange Code). It was a developed form of some of the existing codes like BCD. It is an 8-bit code so 256 different characters can be represented in EBCDIC. It was the most frequently used character code but with the increased use of the personal computer and computer networks the ASCII coding scheme became the standard coding scheme and now most of the computers use ASCII. Following table gives some of the characters and their EBCDIC codes.

Table showing EBCDIC Codes for different characters

Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code	Character
C0	{	D0	}	E0	\	F0	0
C1	A	D1	J	E1		F1	1
C2	B	D2	K	E2	S	F2	2
C3	C	D3	L	E3	T	F3	3
C4	D	D4	M	E4	U	F4	4
C5	E	D5	N	E5	V	F5	5
C6	F	D6	O	E6	W	F6	6
C7	G	D7	P	E7	X	F7	7
C8	H	D8	Q	E8	Y	F8	8
C9	I	D9	R	E9	Z	F9	9

5.7.4 Unicode

Unicode is another popular coding scheme used these days. It is a 16-bit coding scheme so more than $2^{16} = 65536$ characters can be represented in this coding scheme

Exercise

1. Explain the following:
   - **a.** Binary number system
   - **b.** Octal number system
   - **c.** Decimal number system
   - **d.** Hexadecimal number system
   - **e.** ASCII codes
   - **f.** BCD

2. Explain the following terms with examples.
   a. Data
   b. Information
3. What are the main types of data used in different computer applications? Explain the uses of each of the data types and the operations performed on it.
4. Explain the 1's complement method of representing signed numbers. How can you perform subtraction using this method?
5. Explain the 2's complement method of representing signed numbers. How can you perform subtraction using this method?
6. Convert the following decimal numbers into binary, octal and hexadecimal
   a. 78
   b. 97
   c. 129
7. Convert the following hexadecimal numbers into binary, octal and decimal
   a. $7A_{(16)}$
   b. $1C2_{(16)}$
   c. $89_{(16)}$
8. Convert the following octal numbers into binary, decimal and hexadecimal
   a. $125_{(8)}$
   b. $57_{(8)}$
   c. $777_{(8)}$
9. Convert the following binary numbers into octal, decimal and hexadecimal
   a. $01110101_{(2)}$
   b. $10101001_{(2)}$
   c. $00110011_{(2)}$
10. Convert the following BCD numbers into Decimal00111001
   a. 00111001 b. 00000111
   c. 10000001
11. Represent the following numbers as 8-bit 1's complement and 10-bit 2's complement numbers
   a. 76
   b. -98
   c. -126
12. Represent the following 8-bit 1's complement numbers into decimal
   a. 00101011
   b. 10001001
   c. 11111111
13. Represent the following 8-bit 2's complement numbers into decimal
   a. 00111101
   b. 11111111
   c. 10101010
14. Perform the following subtraction using 8-bit 1's complement method. Verify your answer by converting it into decimal. All numbers are in decimal system.
   a. 57 – 126
   b. 120 – 76
   c. -20 - 52
15. Perform the following subtraction using 8-bit 2's complement method. Verify your answer by converting it into decimal. All numbers are in decimal system.
   a. 127 – 126
   b. 12 – 106
   c. -12 – 25

16. **Perform the following subtraction using 8-bit 1's and 2's complement method. Verify the results by converting your answer into decimal. Explain why the results are not correct if there is needed.**

a. -57 – 96
b. -120 – 110
c. - 60- 68

17. **Perform the following subtraction using 10-bit 1's and 2's complement method. Verify the results by converting your answer into decimal.**

a. -57 – 96
b. -120 – 110
c. - 60- 68

18. **What are the smallest and largest numbers that could be represented in 8 bits?**

19. **What are the smallest and largest numbers that could be represented in 8 bit 1's complement form?**

20. **What are the smallest and largest numbers that could be represented in 8 bit 2's complement form?**

21. **Represent the following numbers using fixed point representation. Use the following format for the conversion. Also verify your results by converting your results back into decimal.**

a. 25.5
b. 233.9
c. 33.6

10 bits for integral part	**6 bits for fractional part**

22. Represent the following numbers using fixed point representation. Use the format given in the previous question for the conversion. Explain if there is any trouble.

a. 1025.5
b. -1233.9
c. -2333.6

23. Represent the following numbers using floating point representation.
Use the floating point format given in the chapter

a. 1025.5
b. -1233.9
c. -2333.6

24. **Represent the following messages using the ASCII codes given in the table of ASCII codes. Also verify your coded message by converting it back into English. (Do not forget to convert the space character)**

a. He is a good student
b. 2 + 2 = 4
c. I like Computer Science
d. Binary numbers are GREAT

25. **Fill in the blanks:**

(i) ________________ **is raw facts of knowledge which is ready for processing.**
(ii) **Processed data is called** ________________.
(iii) ______________, ______________, **and** ______________ **are three methods of representing signed numbers.**
(iv) **ASCII stands for** ________________________________.
(v) **1 0000 0100 0010 is equal to** ________________$_{(16)}$.
(vi) **1 000 100 010 is equal to** ________________$_{(8)}$.
(vii) **2's complement of 0010 0011$_{(2)}$ is** ________________.
(viii) **Computer manipulates everything in terms of** ________________.

(ix) Base of hexadecimal numbers is ________.
(x) In ________ end carry is discarded.

26. Match the following:

Data	American Standard Code for Information Interchange
Processing	$22_{(10)}$
Information	Processed Data
ASCII	Raw facts to which no meaning is attached and is ready for processing
$16_{(16)}$	Processing means to manipulate, calculate, distribute or arrange
$12_{(16)}$	$22_{(8)}$

27. Choose the correct answer:

a. The hexadecimal number $10_{(16)}$ is equal to
(i) $10_{(10)}$ (ii) $100_{(10)}$ (iii) $16_{(10)}$ (iv) All of above

b. The hexadecimal number $100_{(16)}$ is equal to
(i) $0001\ 0000\ 0000_{(2)}$ (ii) $256_{(10)}$ (iii) $400_{(8)}$ (iv) All of above

c. 2's complement of $0101010_{(2)}$ is
(i) 1010110 (ii) 1010101 (iii) 0000011 (iv) None of above

d. 1's complement of a negative binary number can be calculated by
(i) reversing the bits in the number
(ii) reversing the bits in the number and adding one
(iii) can not be calculated (iv) both (i)and (ii)

e. (011)4752105 is
(i) numeric data (ii) alphanumeric data (iii) alphabetic data
(iv) both(b)and(c)

28. Mark the following as True/False:

a. It is impossible to build a computer that uses decimal number system
b. $1234_{(16)} = 110\ 11\ 100_{(2)}$ **c.** All computers in the world use ASCII.
d. 1's and 2's complement methods works only for fixed numbers of bits
e. We can not represent 256 using 8 Bits
f. There are total of 8 basic digits in octal number system
g. ASCII is a 7-bit coding scheme
h. Unicode is used to provide multilingual support in software
i. BCD stands for Binary Coded Digits
j. The value of G represents 16 in hexadecimal number system.

Answers

Q.25
(i) Data (ii) Information (iii) signed magnitude, 1's Complement, 2's Complement
(iv) American Standard Code for Information Interchange (v) 1024 (vi) 1024
(vii) 1101 1101 (viii) Binary numbers (ix) 16 (x) 2's complement

Q.27
a. (iii) b. (iv) c. (i) d. (i) e. (ii)

Q.28
a. F b. F c. F d. T e. T f. T g. T h. T i. F j. F